SANDIA REPORT
SAND99–8245
Unlimited Release
Printed March 1999

A Class of Trust-Region Methods for Parallel Optimization

P. D. Hough, J. C. Meza

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

SAND98-8245
Unlimited Release
Printed March 1999

A Class of Trust-Region Methods for Parallel Optimization

P.D. Hough[1], J. C. Meza[2]
Computational Sciences and Mathematics Research Department, MS 9011
Sandia National Laboratories
Livermore, CA 94551-0969

ABSTRACT

We present a new class of optimization methods that incorporates a Parallel Direct Search (PDS) method within a trust-region Newton framework. This approach combines the inherent parallelism of PDS with the rapid and robust convergence properties of Newton methods. Numerical tests have yielded favorable results for both standard test problems and engineering applications. In addition, the new method appears to be more robust in the presence of noisy functions that are inherent in many engineering simulations.

Keywords: parallel optimization, direct search methods, nonlinear programming.

[1]email: pdhough@ca.sandia.gov
[2]email: meza@ca.sandia.gov

1 Introduction

Optimization of functions derived from the modeling and simulation of some physical process constitutes an important class of problems in many engineering and scientific applications. Typically, the computer simulation entails the solution of a system of nonlinear partial differential equations (PDE) in two or three dimensions. Other applications include particle dynamics simulations or problems in chemical kinetics. The main characteristic of these types of problems is that the function evaluation is computationally expensive and dominates the total cost of the optimization problem. Depending on the nature of the application and the solution method employed, there can also be noise associated with the evaluation of the objective function. This noise can usually be reduced, but only at the cost of making the computation time even greater. In many of these applications, derivative information is also not available or must be computed using finite differences, thereby generating noisy gradients. Fortunately, the dimension of the optimization problem in many of these optimal design problems is small to medium (usually on the order of tens of parameters). In this study, we will concentrate on the development of parallel unconstrained optimization algorithms for the solution of these types of problems on small to medium scale shared memory processors (SMP's), where the number of available processors is comparable to the number of optimization parameters. The rationale for this decision is that although massively parallel computers are available, the majority of computational power in most industrial or scientific settings consists of small to medium scale clusters of SMP's or networks of workstations (NOW's) that can be used in a similar capacity.

There have been many attempts at parallelizing nonlinear optimization methods. In the area of Newton methods, one of the earliest attempts at parallelization was the work of Straeter [17], who developed a parallel rank-one updating formula for the Hessian approximations used in variable metric methods. This formula was later extended by Laarhoven [12] to more general updating formulas. Byrd, Schnabel and Shultz [2] also proposed parallel Quasi-Newton methods based on speculative gradient and Hessian evaluations. Schnabel [16] gave an excellent review of the challenges and limitations in parallel optimization. In that review, Schnabel identified three major levels for introducing parallelism: 1) parallelize the function, gradient, and constraint evaluations, 2) parallelize the linear algebra, and 3) parallelize the optimization algorithm at a high level.

In this study, we choose to focus on the third option due to the characteristics of the problems mentioned above. In particular, the first option is not typically available to us because for many situations we do not have access to the source code for the function or the constraints. In addition, the dimension of the optimization problems of interest is usually small, and therefore parallelizing the linear algebra would not yield any benefits.

The third option, that of parallelizing optimization at a high level, has recently received more attention. Some attempts that fit into this category include methods such as parallel direct search methods [7], genetic algorithms [11], and simulated annealing [10]. These methods are inherently parallel and extremely popular in engineering optimization. Although direct search methods can be a powerful tool, they suffer from slow convergence and thus may require many function evaluations. In a setting where each function evaluation may take several CPU hours to compute this is highly undesirable.

Newton-based methods, on the other hand, have good convergence properties, but there are few options for parallelizing a typical Newton method at a high level. For the purposes of this paper, we will assume that the gradient of the objective function is not available and that finite differences are used to compute any necessary derivative information. Since this calculation is easily parallelized, we will not consider this option. Another approach to parallelization is the work by Phua and Zeng [15] in which they use a multiple line search, multiple direction algorithm to introduce parallelism into a Newton method. However, it is not clear how robust a line search method would be in a situation where the function and gradient are noisy. Carter [3] has addressed the issue of inexact gradients for another class of algorithms known as trust-region methods and has given conditions under which these algorithms will converge. In a separate paper, Carter presented various numerical results [4] for this class of algorithms.

In this paper, we consider a new class of methods that combines the parallel direct search method and the trust-region method to produce a new class of algorithms that takes advantage of the best properties of both algorithms. In particular, we will show that the rapid convergence rates typical of Newton type methods is preserved while gaining the advantage of parallelism inherent in the parallel direct search methods. In section 2, we describe the new class of algorithms and consider their convergence properties in section 3. In section 4, we give numerical results from a set of test problems and an application in optimal design. We conclude in section 5 with a summary and a brief discussion of future research directions.

2 The Trust-Region PDS Algorithm

Before describing the new class of algorithms, we first give a brief overview of the standard trust-region method and the parallel direct search (PDS) method due to Dennis and Torczon [7]. In each iteration of a trust-region method, a quadratic model of the objective function, f, is formed, and a region in which the model is trusted to approximate the actual function accurately is determined. A trial step is then computed by solving the following subproblem:

$$\min \quad \psi(\mathbf{s}) = g(\mathbf{x}_c)^T\mathbf{s} + \frac{1}{2}\mathbf{s}^T H_c \mathbf{s} \tag{1}$$

$$\text{s. t.} \quad \|\mathbf{s}\|_2 \leq \delta_c,$$

where $\mathbf{x}_c$ is the current point, $\mathbf{s}$ is the step, $g(\mathbf{x}_c)$, is the gradient of f at the current point, H_c is the current Hessian approximation, and δ_c is the size of the trust region. We will refer to this as the trust-region subproblem. In a typical implementation, the step $\mathbf{s}$ will be a step in the steepest descent direction, the Newton direction, or some convex combination of the two. The trust-region method is illustrated in Figure 1.

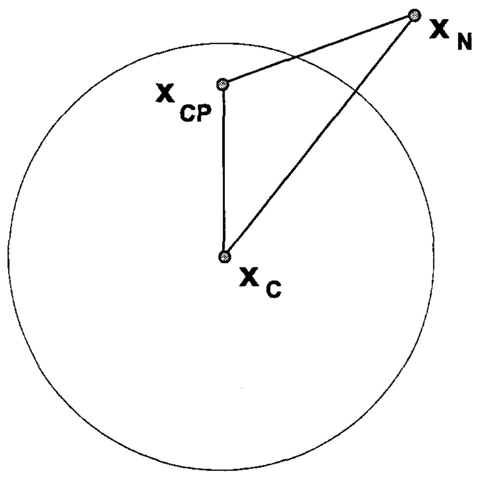

Figure 1: Standard trust-region method

The parallel direct search algorithm belongs to a class of optimization methods that do not compute derivatives. The PDS algorithm can be briefly described as follows. Starting from an initial simplex S_0, the function value at each of the vertices in S_0 is computed and the vertex corresponding to the lowest function value, v_0, is determined. Using an underlying grid structure, S_0 is reflected about v_0 and the function values at the vertices of this rotation simplex, S_r, are compared against the function value at v_0. If one of the vertices in S_r has a function value less than the function value corresponding to v_0, then an expansion step to form a new simplex, S_e, is attempted in which the size of S_r is expanded by some multiple, usually 2. The function values at the vertices of S_e are compared against the lowest function value found in S_r. If a lower function value is encountered, then S_e is accepted as the starting simplex for the next iteration; otherwise S_r is accepted for the next iteration. If no function value lower than the one corresponding to v_0 is found in S_r, then a contraction simplex is created by reducing the size of S_0 by some multiple, usually 1/2, and is accepted for the next iteration.

Because PDS only uses function comparisons, it is easy to implement and use. Since the rotation, expansion, and contraction steps are all well determined, it is possible to precompute a set of grid points corresponding to the vertices of the simplices constructed from various combinations of rotations, expansions, and contractions. Given this set of grid points, called a search scheme, the PDS algorithm can compute the function values at all of these vertices in parallel and determine the vertex corresponding to the lowest function value. The number of points used in the search scheme is referred to as the *search scheme size* and it usually is adjusted to be at least equal to the number of processors available. Figure 2 demonstrates a typical PDS iteration.

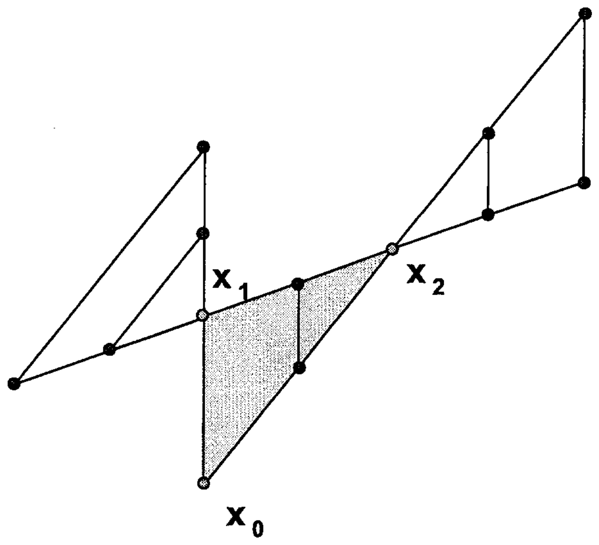

Figure 2: PDS method

Both methods have advantages and disadvantages, as described in Section 1. In order to combine the strengths of these methods, we propose a new class of algorithms which uses the PDS method within a trust-region framework. This type of algorithm, which we will refer to as TRPDS, is illustrated in Figure 3 and is described below. The controlling framework is the same as that for standard trust-region algorithms, but the method of computing the new step is different. Rather than solving the trust-region subproblem, the TRPDS method solves

$$\begin{array}{ll} \min & f(x_c + s) \\ \text{s. t.} & \|s\|_2 \le 2\delta_c. \end{array} \tag{2}$$

We will refer to this as the PDS subproblem. There are a few important points to note. The first is that the actual objective function is being minimized as opposed to a quadratic model of the objective function. Also, this subproblem is not solved to optimality; only a small amount of decrease is required from PDS. Finally, the step length is allowed to be twice the size of the trust region. This is to allow for the possibility of taking twice the Newton step, as is sometimes done in the Newton framework.

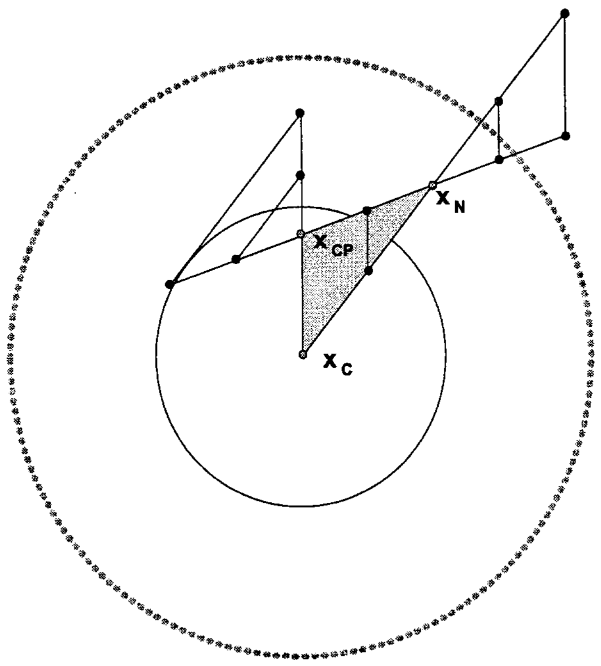

Figure 3: TRPDS method

An overview of the TRPDS algorithm appears below, followed by a discussion of the critical steps.

Algorithm 1. TRPDS

```
Given $\mathbf{x}_0$, $\mathbf{g}_0$, $H_0$, $\delta_0$, and $\eta \in (0,1)$
for $k = 0, 1, \ldots$ until convergence do
  1. Solve $H_k \mathbf{s}_N = -\mathbf{g}_k$
  for $i = 0, 1, \ldots$ until step accepted do
    2. Form an initial simplex using $\mathbf{s}_N$
    3. Find an approximate solution $\mathbf{s}_i$ to (2) using PDS
    4. Compute $\rho_i = (f(\mathbf{x}_k + \mathbf{s}_i) - f(\mathbf{x}_k))/\psi_k(\mathbf{s}_i)$
    if $\rho_i > \eta$ then
      5. Accept step and set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_i$
    else
      6. Reject step
    end if
    7. Update $\delta$
  end for
end for
```

Here we use the notation, $\mathbf{g}_k = g(\mathbf{x}_k)$, and $H_k = H(\mathbf{x}_k)$. There are several points to consider within this framework. The initial simplex formed in Step 2 needs to be chosen carefully. While there is a lot of freedom in the choice of the initial simplex, it will have an impact on the solution to (2) and on the performance of the algorithm. There is also a question as to how accurately we should solve (2). In many applications it may be reasonable to ask for only a small fraction of decrease in the function since each function evaluation is so expensive. This also has a bearing on the decision to accept or reject the new step. Finally, the updating of the trust region must be addressed within the context of this framework. In the following sections, we address each one of these issues.

2.1 Choosing the initial simplex

In Step 2 of the TRPDS algorithm we require the formation of an initial simplex. The choice of this simplex is important to the performance of the PDS algorithm and deserves careful consideration. There are three points that must be included in the simplex: the current point, the Cauchy point, and the Newton point. The Cauchy point is defined to be the minimizer of the quadratic model along the steepest descent direction. Likewise, we define the Newton point, $\mathbf{s}_N$, to be the minimizer of the quadratic model along the Newton direction. The current point must be in the simplex so that PDS can determine whether or not it has found a descent direction. The Cauchy point is required to ensure convergence of the algorithm. Finally, the Newton point is necessary to allow for the possibility of rapid convergence in the limit. In practice, all three of these points are not always included, but instead related points are used. A discussion of these substitutions follows.

Recall that in the standard trust-region algorithm, the step length is limited by the trust region. When the Cauchy point or the Newton point falls outside of the trust region, it is projected onto the trust region. As we are seeking to preserve as much of the trust-region framework as possible, the construction of the initial simplex includes similar features. There are three different scenarios that must be addressed.

1. *Both the Cauchy point and the Newton point are inside the trust region.* This case is straightforward. The Cauchy point and the Newton point are used in the initial simplex.

2. *The Cauchy point is inside the trust region, and the Newton point is outside the trust region.* The Cauchy point is used in the initial simplex. The dogleg point ([6], p. 139) is computed and replaces the Newton point in the initial simplex.

3. *Both the Cauchy point and the Newton point are outside the trust region.* Both points are projected onto the trust region, and the resulting points are used in the initial simplex.

For a problem of dimension n, PDS requires the initial simplex to have $n+1$ vertices. We have described the selection of only three. The question of how to pick the remaining $n-2$ vertices remains. While there are many logical ways to choose these points, the only real restriction on them is that they be chosen such that the initial simplex is not degenerate. Our current implementation uses $n-2$ vertices from a right angle simplex. This simplex can be generated by defining the remaining vertices to be a fixed distance in each of the n coordinate directions from the Newton point (or its projection). The distance from each of these vertices to the Newton point is the same as the radius of the trust region.

By constructing the simplex in the manner described here, there are two situations in which it may be degenerate. One case can arise in the first iteration of the trust-region method. If the initial Hessian is a multiple of the identity, then the Newton direction and the Cauchy direction will be the same, so the simplex needs to be constructed in a slightly different manner in the first iteration. We use the current point, the Newton point, and $n-1$ of the vertices from the right angle simplex around the Newton point. The distance between the Newton point and the remaining points is the same as the radius of the trust region. The other case of degeneracy arises if the edges of the simplex are badly scaled. This is easily corrected by re-scaling all of the edges to be the same length as the Newton edge. Note that this allows longer steps than if all edges were re-scaled to be the same length as the Cauchy edge.

2.2 Solving the PDS subproblem

One way to think about the TRPDS algorithm is to imagine using an optimization algorithm within an optimization algorithm. As such, the PDS method needs algorithmic parameters in order to solve the PDS subproblem. The initial simplex just described is only one of these parameters. The PDS method evaluates the function at a set of pre-determined reflection, contraction, and expansion points in order to determine a trial step. Recall, that the search scheme for the PDS method can be determined ahead of time and need only be generated once. However, during the optimization phase, the PDS method must determine how many points in the search scheme to evaluate at each iteration. One possible choice in a parallel setting, is to set this number equal to the number of processors that are available. PDS also needs to know the size of the trust region, as that constrains the size of the step. In practice, we relax the trust-region constraint in order to allow for the possibility of taking twice the Newton step. Another constraint is that the step taken must satisfy a fraction of Cauchy decrease condition according to some model; the implications of using various models are considered in section 3.

Since the PDS subproblem is not solved to optimality, we use three criteria to determine when to return a new step. The first is a simple decrease requirement. If a point with a function value that is less than some large fraction of the function value at the current point is found, then PDS returns the associated step. If such a step is not found, then PDS returns when it has exceeded either the maximum number of function evaluations or the maximum number of iterations allowed. In our implementation, we set the maximum number of function evaluations to be the same as the search scheme size. This usually limits PDS to one iteration. However, vertices of the simplex that are outside of the trust region are not evaluated (because they are infeasible); thefore PDS may require additional iterations to reach the maximum number of function evaluations so we also limit the number of iterations. Currently, we use an upper bound of 1-3 iterations, though we find that this bound is not usually needed in practice.

2.3 Acceptance/Rejection of step

Once a step has been computed, it is necessary to determine whether or not it is acceptable. This is handled by the trust-region framework. If there is sufficient decrease, then the step will be accepted. Otherwise, the step is rejected. In a standard trust-region method, sufficient decrease is determined by computing ρ_k as given in Step 4 of the algorithm and comparing it to some tolerance that depends on the computational expense of the function. If it is greater than the tolerance, then the decrease is sufficient. There are two situations that arise in the TRPDS algorithm that require minor modifications to this scheme. It is possible that PDS will find no decrease, and thus return a step of zero length. In this case, ρ_k is not computed, and the step is rejected immediately. Because PDS minimizes the actual function and not the quadratic model, it may compute a step for which the model predicts increase. Again, ρ_k is not computed, but the step is accepted since there is reduction in the actual function.

2.4 Updating the trust region

The procedure for updating the trust region is based on the value of ρ and on the length of the computed step. At each iteration, the trust region is updated as follows:

$$\delta_{k+1} = \begin{cases} \min(\delta_k, \|E\|_2)/10, & \text{if } \rho_k < \eta_1 \text{ or } \|\mathbf{s}_k\|_2 = 0 \\ \|\mathbf{s}_k\|_2/2, & \text{if } \eta_1 \leq \rho_k \leq \eta_2 \\ 2 \cdot \delta_k, & \text{if } \eta_3 \leq \rho_k \leq 2 - \eta_3 \\ \max(2 \cdot \|\mathbf{s}_k\|_2, \delta_k), & \text{otherwise.} \end{cases}$$

Here E is the longest edge of the final PDS simplex, and $\eta_1, \eta_2, \eta_3 \in (0,1)$. Notice that when the trust-region size is reduced, the radius of the new trust region may not be any larger than E, thus eliminating the possibility of re-checking points that are already known to be unacceptable. In our algorithm, we also impose a maximum trust region that is allowed at any iteration.

3 Convergence Results

Since our new algorithm can be viewed as a special case of a trust region algorithm the standard convergence results can be applied. However, it is interesting to note that Algorithm 1 can be viewed as a specific instance of the more general framework for approximation models described in [1]. Within that framework the PDS subproblem (2) is a subcase of the approximation models proposed in that report. As such, if we make the standard assumptions used for the classical trust-region algorithms and the approximation model satisfies two additional conditions described in [1], then convergence for the new algorithm is easy to prove.

In particular, we must assume that the sequence of iterates satisfies a condition commonly known as fraction of Cauchy decrease. That is, there exist constants, $\beta > 0$ and $C > 0$, independent of k, for which the step s_k taken at iteration k satisfies

$$f(\mathbf{x}_k) - a_k(\mathbf{x}_k + \mathbf{s}_k) \geq \beta \|g(\mathbf{x}_k)\|_2 \min\left(\delta_k, \frac{\|g(\mathbf{x}_k)\|_2}{C}\right), \tag{3}$$

where a_k denotes the model being used to approximate the objective function. In addition, the approximation model, a_k, used within the trust-region framework must also satisfy the following two conditions:

$$\begin{aligned} a_k(\mathbf{x}_k) &= f(\mathbf{x}_k), & (4)\\ \nabla a_k(\mathbf{x}_k) &= \nabla f(\mathbf{x}_k). & (5) \end{aligned}$$

Theorem 3.1 *Assume that f is uniformly continuously differentiable, bounded below, and the Hessian approximations are uniformly bounded. Furthermore assume that the sequence of iterates generated by Algorithm 1 satisfies (3)-(5). Then*

$$\lim_{k\to\infty} \inf \|\nabla f(\mathbf{x}_k)\|_2 = 0.$$

In our particular case, the approximation model is exactly the objective function that we are seeking to minimize. Using the terminology in [1] this would correspond to a second-order model. Note that we are only using function information to solve the PDS subproblem and hence we do not take advantage of any first or second order information inherent in the model. However, this information is used outside the subproblem to compute the initial Newton step and Cauchy point. Thus, this method has characteristics of both zero-order and second-order methods.

In fact, since Algorithm 1 uses widely accepted criteria for updating the steps within the trust-region framework, it is easy to show that the stronger condition

$$\lim_{k\to\infty} \|\nabla f(\mathbf{x}_k)\|_2 = 0$$

holds.

This is satisfying, in that the new class of algorithms inherits all of the good theoretical convergence properties of the classical trust region methods while incorporating all of the practical advantages of PDS for typical engineering optimization problems.

4 Numerical Results

In order to evaluate the performance of the TRPDS algorithm in practice, we chose a set of test problems from the literature. These problems were obtained from papers by Moré, Garbow, and Hillstrom [14], Byrd, Schnabel, and Shultz [2], and Conn, Gould, and Toint [5]. Some of these problems are also used by Phua and Zeng [15]. For comparison purposes, we also solved these problems using a standard BFGS trust-region algorithm.

The starting points used for these problems were the same as those given in the references. Computation of finite difference gradients is currently not parallelized. All initial conditions are listed below:

$$\begin{aligned}
\text{Initial Trust Region} &= 0.1 \cdot \|\mathbf{g}_0\|_2 \\
\text{Machine Epsilon} &= 2.22045 \cdot 10^{-16} \\
\text{Maximum Step} &= 4000 \\
\text{Minimum Step} &= 1.49012 \cdot 10^{-8} \\
\text{Maximum Iter} &= 500 \\
\text{Maximum Fcn Eval} &= 1,000,000 \\
\text{Step Tolerance} &= 1.49012 \cdot 10^{-8} \\
\text{Function Tolerance} &= 1.49012 \cdot 10^{-8} \\
\text{Gradient Tolerance} &= 10^{-6}
\end{aligned}$$

The step tolerance, the function tolerance, and the gradient tolerance are used as stopping criteria for the optimization algorithms. At the end of each iteration the step length relative to the size of the current point, the change in the function relative to the current function value, and the size of the gradient relative to the current function value are determined. When one of these falls below the associated tolerance, the algorithm has converged.

The results of the experiments appear in Tables 1–3. While reporting the number of iterations taken may be useful in some settings, we believe that it is not useful here. In our applications, the most important measure of performance is the total time to solution of the problem. Since the computational cost of the function evaluations dominates the cost of the algorithm, we base our comparison of TRPDS with BFGS on the number of function evaluations. Since the TRPDS algorithm is run in parallel and the trust-region method is serial, comparing the total number of function evaluations required for each is not a fair method of comparison.

Instead, we compare the number of *concurrent function evaluations* defined as follows. Suppose that p processors are available, where $p \geq 1$. If p independent function evaluations are required, then each processor can be tasked to perform one of them. This means that all function evaluations can be done simultaneously. Thus, we define a *concurrent function evaluation* to be one instance of p function evaluations being performed simultaneously. Comparing the number of concurrent function evaluations is therefore roughly equivalent to comparing total wall clock time. The numbers that appear in Tables 1–3 are the ratios of the number of concurrent function evaluations taken by the TRPDS algorithm to the number of concurrent function evaluations taken by the serial trust region method. Each table corresponds to a different dimension for the problem set (represented by n). Ratios less than one indicate that the TRPDS algorithm will take less overall time.

The tests were run on a 64-processor SGI Origin 2000 with the IRIX 6.4 operating system. For the TRPDS algorithm, the ideal situation is to have the same number of search scheme points as processors so that each processor evaluates the function at one search scheme point per iteration. However, with limitations on the available resources, this was not always possible. In particular, in the 16-dimensional problems with a search scheme size of 128, each processor evaluated two search scheme points per iteration.

Table 1: Ratios of concurrent function evaluations for the TRPDS algorithm versus a standard BFGS trust-region method.

$n = 4$	**Search Scheme Size**			
problem	$n + 1$	$2 * n$	$4 * n$	$8 * n$
broyden1a	0.72	0.72	0.72	0.51
broyden1b	1.18	1.07	1.07	1.07
broyden2a	0.66	0.72	0.59	0.59
broyden2b	0.62	0.62	0.62	0.55
chain_singular	0.46	0.62	0.64	0.37
chebyquad	1.50	1.02	1.27	1.44
cragg_levy	0.93	0.68	0.83	0.79
epowell	0.93	0.93	0.98	0.93
erosen	1.54	1.45	1.31	1.23
gen_brown	1.06	1.06	0.94	1.06
penalty1	0.88	0.88	0.48	0.48
penalty2	1.18	1.18	1.18	1.18
toint_trig	1.96	2.84	2.32	2.24
tointbroy	1.08	1.08	1.08	0.88
trig	1.03	0.95	0.95	0.87
vardim	0.50	0.50	0.50	0.76

Table 2: Ratios of concurrent function evaluations for the TRPDS algorithm versus a standard BFGS trust-region method.

$n = 8$	**Search Scheme Size**			
problem	$n + 1$	$2 * n$	$4 * n$	$8 * n$
broyden1a	0.86	0.77	0.82	0.72
broyden1b	1.02	0.95	0.95	1.02
broyden2a	0.38	0.38	0.34	0.34
broyden2b	0.55	0.55	0.55	0.55
chain_singular	0.47	0.49	0.48	0.35
chebyquad	1.01	0.92	0.92	0.99
cragg_levy	0.40	0.36	0.42	0.55
epowell	0.89	0.89	0.97	0.89
erosen	0.81	1.40	1.90	1.86
gen_brown	0.93	0.85	0.85	0.68
penalty1	1.12	2.30	2.07	0.45
penalty2	0.95	0.67	0.85	0.71
toint_trig	2.43	2.38	1.97	2.39
tointbroy	0.81	0.81	0.96	1.03
trig	0.80	0.80	0.72	0.69
vardim	0.64	1.40	1.51	1.28

Table 3: Ratios of concurrent function evaluations for the TRPDS algorithm versus a standard BFGS trust-region method.

$n = 16$	**Search Scheme Size**			
problem	$n+1$	$2*n$	$4*n$	$8*n$
broyden1a	0.66	0.73	0.66	0.73
broyden1b	1.06	1.00	1.06	1.05
broyden2a	0.36	0.36	0.33	0.32
broyden2b	0.51	0.51	0.47	0.61
chain_singular	0.45	0.44	0.35	0.37
chebyquad	0.81	0.76	0.70	0.89
cragg_levy	0.46	0.42	0.59	0.39
epowell	0.83	0.79	0.75	0.79
erosen	0.81	2.61	1.19	0.85
gen_brown	0.58	0.64	0.53	0.55
penalty1	0.55	0.55	1.11	0.42
penalty2	0.45	0.50	0.37	0.34
toint_trig	1.94	0.93	0.93	0.97
tointbroy	0.75	0.75	0.90	0.90
trig	0.73	0.73	0.68	0.66
vardim	0.84	0.84	1.14	1.02

The results shown in Tables 1–3 allow us to make some general observations about the effect of the search scheme size and the dimension of the problem. For a given problem, the size of the search scheme has very little, if any, effect on the performance of the algorithm in comparison to the standard trust region algorithm. The dimension of the problem appears to have an effect in some cases, but there is no clear trend.

Schnabel [16] presents an argument for the merits of a line search algorithm with speculative gradient evaluation. This entails using extra processors to compute the components of a finite difference gradient at the current point while the function is being evaluated at that point. In essence, he argues that it is difficult to develop a parallel line search algorithm that will perform better than a speculative gradient algorithm, particularly when the dimension of the problem is not much larger than the number of available processors. We present a theoretical comparison between our algorithm and a speculative gradient algorithm based on the analysis presented by Schnabel. The number of concurrent function evaluations required by the speculative gradient algorithm is given by

$$\left(\gamma_s + \left\lceil \frac{n+1}{p} \right\rceil\right) * It_s,$$

where n is the dimension of the problem, p is the number of processors, and It_s is the number of iterations. So $\lceil \frac{n+1}{p} \rceil$ is the number of concurrent function evaluations required to compute the function and the gradient at a successful trial point. The variable γ_s denotes the average number of unsuccessful trial points per iteration. Note that rejecting γ_s points per iteration corresponds to performing γ_s concurrent function evaluations per iteration. For the TRPDS algorithm, each iteration requires $\lceil \frac{n}{p} \rceil$ concurrent function evaluations for the gradient, $\lceil \frac{n}{p} \rceil$ for the initial simplex, and $\lceil \frac{SSS}{p} \rceil$ for the search scheme points, where SSS is the search scheme size. Therefore, the total number of concurrent function evaluations is given by

$$\left(2 * \gamma_{npds} + 2 * \left\lceil \frac{n}{p} \right\rceil + \left\lceil \frac{SSS}{p} \right\rceil\right) * It_{npds}.$$

Note that we are assuming that the PDS algorithm takes one iteration. Also, it is necessary to add $2 * \gamma_{npds}$. This is because rejecting one step in the TRPDS algorithm corresponds to performing two additional concurrent function evaluations, one for the initial simplex and one for the search scheme. In order for the TRPDS algorithm to beat the speculative gradient algorithm, it must take fewer concurrent function evaluations, and thus the following must be true:

$$\frac{It_{npds}}{It_s} < \frac{\gamma_s + \lceil \frac{n+1}{p} \rceil}{2 * \gamma_{npds} + 2 * \lceil \frac{n}{p} \rceil + \lceil \frac{SSS}{p} \rceil}. \tag{6}$$

Tables 4–6 show the ratios of the number of iterations taken by the TRPDS algorithm to the number taken by the standard trust-region method. The contours in Figures 4–6 represent break-even points, as given by the right-hand side of 6, for various values of p and SSS. For the purpose of the figures, $\gamma_s = \gamma_{npds} = 0$. By using the tables and figures together, it is possible to determine for which tests the TRPDS algorithm will be competitive with speculative gradient. For example, consider problem *vardim* with $n = 4$. When $SSS = 5$, Table 4 shows that the TRPDS algorithm took 38% of the iterations required for the standard trust-region method. Now turning our attention to Figure 4, we can compare this number to various break-even points. When $p = 5$, the break-even point is 33%, so a speculative gradient algorithm would be faster than TRPDS. However when $p = 4$, the break-even point is 50%, and thus the TRPDS algorithm would be faster than speculative gradients, in that case.

Table 4: Ratio of number of nonlinear iterations for the TRPDS algorithm versus the trust-region method.

$n = 4$	**Search Scheme Size**			
problem	$n+1$	$2*n$	$4*n$	$8*n$
broyden1a	0.59	0.59	0.59	0.41
broyden1b	1.00	0.90	0.90	0.90
broyden2a	0.53	0.59	0.47	0.47
broyden2b	0.50	0.50	0.50	0.44
chain_singular	0.36	0.50	0.52	0.30
chebyquad	1.25	0.88	1.13	1.25
cragg_levy	0.78	0.56	0.69	0.66
epowell	0.78	0.78	0.83	0.78
erosen	1.30	1.23	1.10	1.03
gen_brown	0.89	0.89	0.78	0.89
penalty1	0.73	0.73	0.36	0.36
penalty2	1.00	1.00	1.00	1.00
toint_trig	1.56	2.33	1.89	1.67
tointbroy	0.91	0.91	0.91	0.73
trig	0.86	0.79	0.79	0.71
vardim	0.38	0.38	0.38	0.62

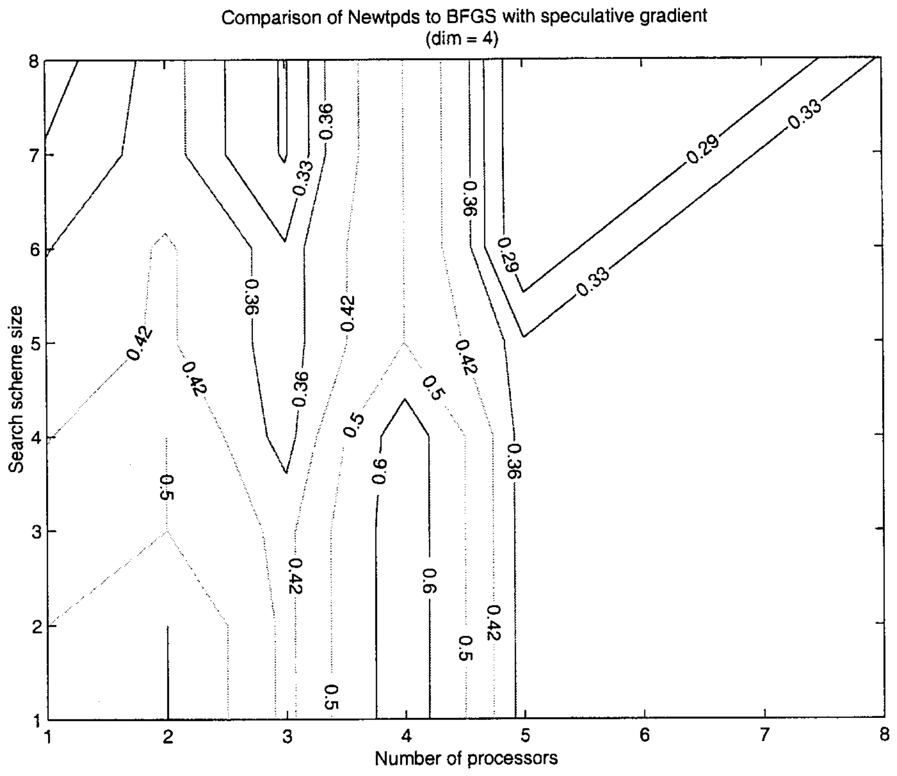

Figure 4: Contour plot of break-even points for dim = 4

Table 5: Ratio of number of nonlinear iterations for the TRPDS algorithm versus the trust-region method.

$n = 8$	**Search Scheme Size**			
problem	$n + 1$	$2 * n$	$4 * n$	$8 * n$
broyden1a	0.77	0.68	0.73	0.64
broyden1b	0.93	0.86	0.86	0.93
broyden2a	0.32	0.32	0.28	0.28
broyden2b	0.48	0.48	0.48	0.48
chain_singular	0.42	0.44	0.43	0.31
chebyquad	0.92	0.85	0.85	0.88
cragg_levy	0.35	0.32	0.37	0.49
epowell	0.80	0.80	0.88	0.80
erosen	0.74	1.28	1.72	1.70
gen_brown	0.83	0.75	0.75	0.58
penalty1	1.00	2.13	1.88	0.38
penalty2	0.84	0.58	0.74	0.63
toint_trig	2.25	2.17	1.75	2.00
tointbroy	0.71	0.71	0.86	0.93
trig	0.71	0.71	0.64	0.61
vardim	0.56	1.28	1.39	1.17

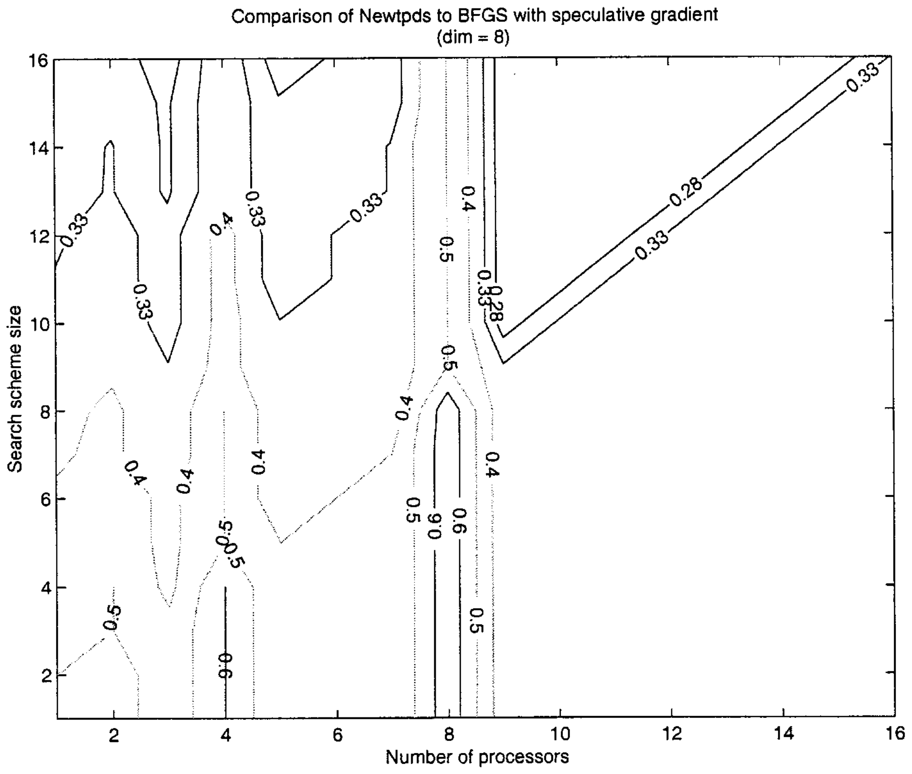

Figure 5: Contour plot of break-even points for dim = 8

Table 6: Ratio of number of nonlinear iterations for the TRPDS algorithm versus the trust-region method.

$n = 16$	**Search Scheme Size**			
problem	$n + 1$	$2 * n$	$4 * n$	$8 * n$
broyden1a	0.61	0.68	0.61	0.65
broyden1b	1.00	0.94	1.00	0.94
broyden2a	0.32	0.32	0.29	0.26
broyden2b	0.46	0.46	0.43	0.54
chain_singular	0.43	0.41	0.33	0.33
chebyquad	0.77	0.72	0.66	0.77
cragg_levy	0.43	0.39	0.56	0.34
epowell	0.78	0.74	0.59	0.70
erosen	0.76	2.51	1.14	0.76
gen_brown	0.53	0.59	0.47	0.47
penalty1	0.50	0.50	1.05	0.35
penalty2	0.41	0.46	0.34	0.29
toint_trig	1.81	0.88	0.88	0.88
tointbroy	0.70	0.70	0.85	0.80
trig	0.68	0.68	0.63	0.58
vardim	0.78	0.78	1.09	0.91

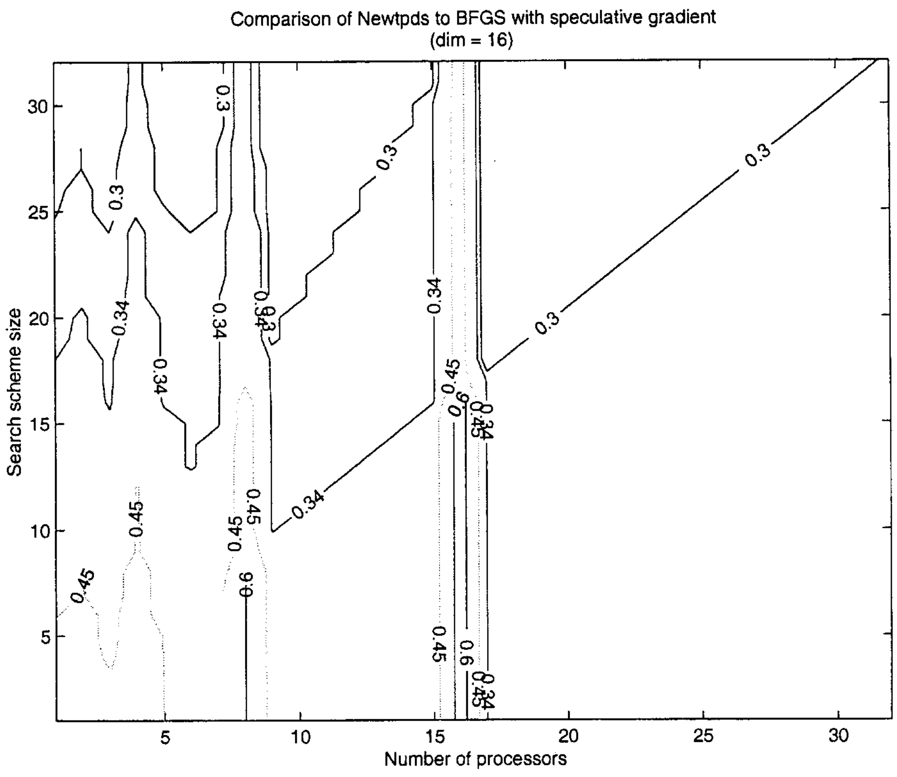

Figure 6: Contour plot of break-even points for dim = 16

Based on these results, we can draw several conclusions. Again, we notice that the size of the search scheme has little effect on the performance of the TRPDS algorithm. There are several factors that may be contributing to this effect. The first is that we have biased the search directions towards the Newton point through our method for constructing the initial simplex. The second factor is the order in which the search scheme points are chosen and evaluated. In the standard PDS algorithm, the reflection points are evaluated first, followed by the contraction points and the expansion points. Since the Newton point is often a good trial point we would not expect nearby points to have a lower function value and that we need to take a large step before finding a better point. Due to the construction of the search pattern this requires a large search scheme before we start to evaluate the expansion points.

In a situation where we are far away from the solution or the function or gradient is noisy, the Newton point might not be a good trial point. In this case, the reflection and contractions points may yield better trial points than the Newton point and thus increasing the search scheme size might improve the performance of the algorithm. This is a point that will require further investigation. It would also be interesting to try different methods for creating the initial simplex to determine the effect on the algorithm's efficiency.

Figures 4–6 indicate that our algorithm is within striking distance of the speculative gradient algorithm when the number of processors is equal to the dimension of the problem and when the search scheme size is less than or equal to the number of processors. Another point to note is that when the number of processors is twice the dimension, a speculative gradient technique could be implemented within the TRPDS algorithm. To summarize, the TRPDS algorithm can be competitive with the speculative gradient algorithm in two cases:

1. $SSS \leq p = n$,
2. $p > 2 * n$.

4.1 Furnace design test problem

As another test case, we chose an optimization problem derived from a design problem of a vertical, multi-wafer furnace. Vertical furnaces can process up to 200 silicon wafers in a single batch and have been used for thin film deposition, oxidation, and other thermal process steps. The evolution of vertical furnaces has been driven by the need for process uniformity (that is, wafer-to-wafer and within-wafer uniformity) and high wafer throughput. A recent variation of the multiwafer reactor design is the small-batch, fast-ramp (SBFR) furnace. The SBFR is designed to heat-up and cool-down quickly, thus reducing cycle time and thermal budget. The SBFR consists of a stack of eight-inch diameter silicon wafers enclosed in a vacuum-bearing quartz jar. The stack is radiatively heated by resistive coil heaters contained in an insulated canister. The heating coils can be individually controlled or ganged together in zones to vary the emitted power along the length of the reactor; a seven-zone configuration is shown in Figure 7. There are six control zones (each containing several heating coils) along the length of the furnace and one heater zone in the base. The zones near the ends of the furnace are usually run hotter than the middle zones to make up for heat loss.

The thermal design optimization problem can be described as follows. Given a set number of fixed heating coils, how can the coils be grouped in the fewest number of control zones such that the temperature deviation about a fixed set-point is minimized. For this example, we concentrate on finding the optimal power settings and related temperature uniformity for a fixed zone configuration. The objective function, F, is defined by a least-squares fit of the N discrete wafer temperatures, $T_{w,i}$, to a prescribed profile, $T_{s,i}$,

$$F(p_j) = \sum_{i=1}^{N} \left(T_{w,i} - T_{s,i}\right)^2, \qquad (7)$$

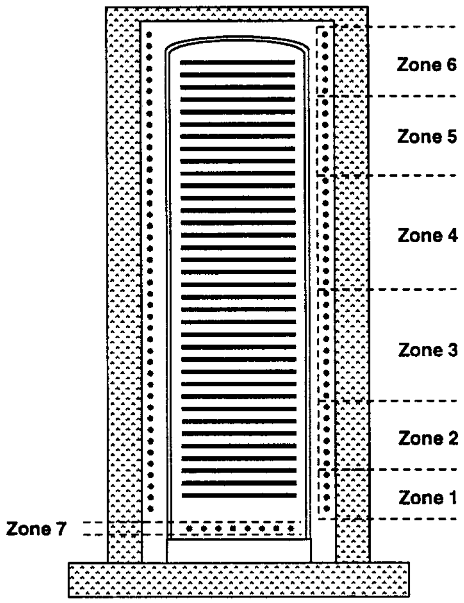

Figure 7: Vertical Batch Furnace with Seven Control Zones

where the p_j are the unknown power parameters.

The engineering heat transfer model used in this example was developed by Houf [9] specifically for the analysis of vertical furnaces (the actual simulation code used in our experiments is called TWAFER). Given a set of powers, p_j, each call to TWAFER produces a set of temperatures for the entire furnace from which the wafer temperatures are extracted. The heat transfer formulation is simplified by using mass lumping and one-dimensional approximations. The nonlinear transport equations are solved using the TWOPNT solver [8], which uses a Newton method with a time evolution feature that computes the steady state solution. By varying the tolerances for the TWOPNT solver it is possible to increase the accuracy of the steady-state solution at the cost of increasing the computational time. In particular, we have chosen to vary a parameter that determines the relative convergence tolerance, RTOL, for the steady-state solution of the underlying PDE.

There are many different parameter combinations that have been considered in previous studies of the TWAFER code [13]. For this particular example we used only one configuration, namely a design problem with 7 heater zones: one bottom heater and six equally-sized side heaters. Each simulation used a model that contained 100 wafers with ten discretization points per wafer. Our initial guess for the powers was: $p_0 = \{100, 200, 300, 2700, 100, 400, 2000\}$.

Table 7 contains the total number of iterations as well as the total wall clock time by each method to compute a solution. With respect to the total wall clock time, the new method is competitive with the standard BFGS method in almost all cases. In addition, the new method is more robust for the larger values of RTOL. These values correspond to a very loose convergence tolerance for the PDE solver in the TWAFER modeling code. In these cases, the standard BFGS method did not converge to a solution while the new method still manages to proceed. In these cases, the algorithm terminated with the trust region shrinking below its minimum allowed size. This failure to converge is probably due to large inaccuracies in the gradient evaluations due to the loose tolerances in the PDE solutions.

Table 7: Number of nonlinear iterations and wall clock time.

	BFGS		NEWTPDS	
RTOL	iter	time	iter	time
10^{-2}	3*	154.471	100	5217.98
10^{-3}	30*	1603.75	100	6872.32
10^{-4}	64	4470.97	100	8857.03
10^{-5}	31	2729.82	25	2786.17
10^{-6}	39	4145.88	33	4468.22
10^{-7}	34	4516.57	24	3832.82
10^{-8}	31	4612.38	26	4548.66
10^{-9}	31	5152.05	25	4969.36
10^{-10}	31	6044.57	22	5101.78
10^{-11}	31	6576.52	28	6915.97
10^{-12}	31	6600.24	23	5774.78
$*$ indicates the method did not converge				

The resulting power values were then given to the TWAFER simulation using a value of RTOL = 10^{-12}. Figures 8-9 show the wafer temperatures that result with the computed powers. The interesting point here is that even with relatively large values of the parameter RTOL the resulting temperatures are still quite reasonable. As we have already noted, for these same values of RTOL the BFGS algorithm did not converge.

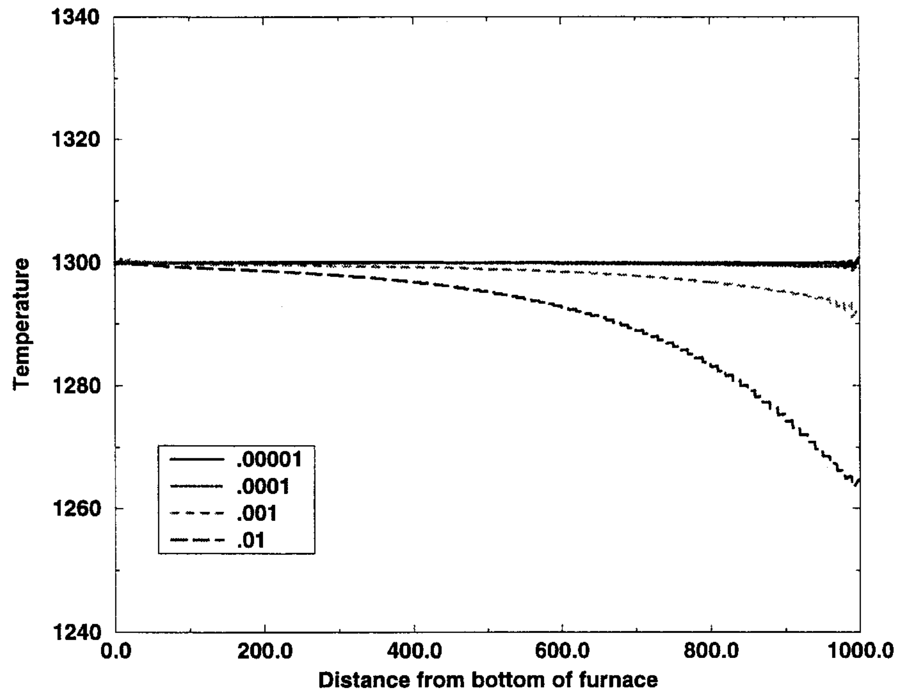

Figure 8: Computed temperatures for various values of RTOL using NEWTPDS

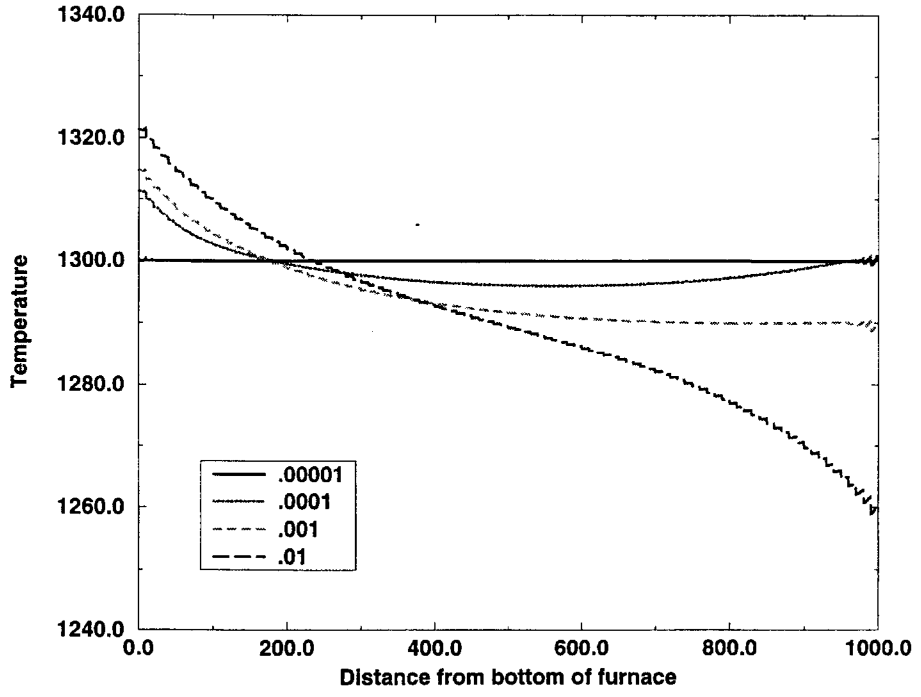

Figure 9: Computed temperatures for various values of RTOL using BFGS

5 Summary

We have described a new class of algorithms for parallel optimization. The general framework consists of a trust-region model where the standard subproblem is solved using a parallel direct search method that takes advantage of parallelism to solve the problem more efficiently on multiprocessors. This new algorithm can be shown to have the same convergence properties as trust-region methods. In addition, the practical properties of the parallel direct search methods can be used to solve engineering optimization problems that are noisy, or lack analytic derivatives.

This new class of algorithms was tested on a standard set of test problems where it performed favorably against the traditional BFGS method. We also tested this new algorithm on a test case derived from an optimal design problem for a chemical vapor deposition furnace. The results indicate that the new method is competitive with the traditional BFGS method. In addition, the new method is more robust in the presence of noise that is generated by the use of less accurate PDE solvers. This is an important feature since many users would prefer to use less accurate PDE solvers in order to reduce the total computational time.

There are many new options to explore. In particular, it would be useful to develop strategies for bound constrained and general inequality constrained problems. We also need to address some issues related to the distribution of function evaluations in order to improve the efficiency of the algorithm. It would also be interesting to explore more general approximation models within this new framework.

Acknowledgments. We wish to thank J.E. Dennis, Jr. and M. Heinkenschloss for many helpful discussions and for pointing out the relationship between our new algorithm and the framework for approximation models.

References

[1] N. Alexandrov, J. E. Dennis, Jr., R. M. Lewis, and V. Torczon, *A trust region framework for managing the use of approximation models in optimization*, Tech. Report 97-50, ICASE, Hampton, VA, 1997.

[2] R. H. Byrd, R. B. Schnabel, and G. A. Shultz, *Parallel quasi-newton methods for unconstrained optimization*, Mathematical Programming, 42 (1988), pp. 273–306.

[3] R. G. Carter, *On the global convergence of trust region algorithms using inexact gradient information*, SIAM J. Numer. Anal., 28 (1991), pp. 251–265.

[4] ——, *Numerical experience with a class of algorithms for nonlinear optimization using inexact function and gradient information*, SIAM J. Sci. Comput., 14 (1993), pp. 368–388.

[5] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, Tech. Report Research Report CS-86-45, University of Waterloo, Waterloo, CA, 1986.

[6] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[7] J. E. Dennis, Jr. and V. Torczon, *Direct search methods on parallel machines*, SIAM J. Opt., 1 (1991), pp. 448–474.

[8] J. F. Grcar, *"TWOPNT Program for Boundary Value Problems, Version 3.10"*, Tech. Report SAND91–8230, Sandia National Laboratory, Livermore, CA, April 1992.

[9] W. G. Houf, J. F. Grcar, and W. G. Breiland, *"A Model for Low Pressure Chemical Vapor Deposition in a Hot-Wall Tubular Reactor"*, Materials Science Engineering, B, Solid State Materials for Advanced Technology, **17** (1993), pp. 163–171.

[10] L. Ingber, *Simulated annealing: practice versus theory*, Mathematical Computer Modeling, 18 (1993), pp. 29–57.

[11] P. Jog, J. Suh, and D. Van Gucht, *Parallel genetic algorithms applied to the traveling salesman problem*, SIAM J. Opt., 1 (1991), pp. 515–529.

[12] P. J. M. Laarhoven, *Parallel variable metric algorithms for unconstrained optimization*, Mathematical Programming, 33 (1985), pp. 68–81.

[13] C. D. Moen, P. A. Spence, and J. C. Meza, *Automatic differentiation for gradient-based optimization of radiatively heated microelectronics manufacturing equipment*, in Proceedings of 6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA., September 1996.

[14] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *Testing unconstrained optimization software*, ACM Transactions on Mathematical Software, 7 (1981), pp. 17–41.

[15] P. K. Phua and Y. Zeng, *Parallel quasi-newton algorithms for large scale optimization.* Web report, 1995.

[16] R. B. Schnabel, *A view of the limitations, opportunities, and challenges in parallel nonlinear optimization*, Parallel Computing, 21 (1995), pp. 875–905.

[17] T. A. Straeter, *A parallel variable metric optimization algorithm*, Tech. Report NASA TN D-7329, NASA, Langley Research Center, Hampton, VA, 1973.

DISTRIBUTION:

1 Dr. Richard H. Byrd
University of Colorado
Department of Computer Science
Campus Box 430
Boulder, CO 80309

1 Dr. Thomas F. Coleman
Cornell University
Department of Computer Science
725 ETC
Ithaca, NY 14853-7501

1 Dr. Edward J. Dean
University of Houston
Department of Mathematics
4800 Calhoun Road
Houston, TX 77004-2601

1 Dr. John E. Dennis, Jr.
Department of Mathematical Sciences
MS 134
Rice University
P. O. Box 1892
Houston, TX 77251-1892

1 Dr. Matthias Heinkenschloss
Department of Mathematical Sciences
MS 134
Rice University
P. O. Box 1892
Houston, TX 77251-1892

1 Dr. Jorge More
Argonne National Laboratories
Mathematical and Computer
Science Division
Argonne, IL 60439-4803

1 Dr. Stephen Nash
Operations Research and Applied Statistics
George Mason University
Fairfax, VA 22030-4444

1 Dr. Jorge Nocedal
Department of Electrical Engineering and Computer Science
Northwestern University
Evanstown, IL 60208-0001

1 Dr. Linda R. Petzold
Department of Mechanical and Environmental Engineering
University of California
Santa Barbara, CA 93106

1 Dr. Robert Schnabel
Department of Computer Science
University of Colorado at Boulder
Campus Box 430
Boulder, CO 80309-0430

1 Dr. William Symes
Department of Mathematical Sciences
MS 134
Rice University
P.O. Box 1892
Houston, TX 77251-1892

1 Dr. Richard Tapia
Dept. of Mathematical Sciences
MS 134
Rice University
P.O. Box 1892
Houston, TX 77251-1892

1 Dr. Virginia Torczon
Department of Computer Science
College of William and Mary
P. O. Box 8795
Williamsburg, VA 23187

1 Dr. Margaret Wright
AT&T Laboratories
Room 2C-462
600 Mountain Avenue
Murray Hill, NJ 07974

1 Dr. Stephen Wright
Argonne National Laboratory
MCS Division
Argonne, IL 60439

1 Stephen A. Vavasis
Department of Computer Science
493 Rhodes Hall
Cornell University
Ithaca, NY 14853

1 Stephen A. Wirkus
Center for Applied Mathematics
657 Rhodes Hall
Cornell University
Ithaca, NY 14853

1 Thomas R. Nicely
Department of Mathematics
Lynchburg College
1501 Lakeside Drive
Lynchburg, VA 24501

1 MS 9001 T. O. Hunter, 8000
Attn: J. B. Wright, 2200
D.L. Crawford, 5200
M.E. John, 8100
L.A. West, 8200
W.J. McLean, 8300
P.N. Smith, 8500
P.E. Brewer, 8600
T.M. Dyer, 8700
L.A. Hiles, 8800

1 MS 9003 D. R. Henson, 8909
1 MS 9007 R. C. Wayne, 8900
1 MS 9011 P. W. Dean, 8903
1 MS 9011 B. Hess, 8910

1	MS 9011	J. C. Meza, 8950
1	MS 9011	M. Rogers, 8960
1	MS 9011	J. A. Larson, 8970
1	MS 9012	K. Hughes, 8990
1	MS 9012	S. Gray, 8930
1	MS 9019	B. A. Maxwell, 8940
1	MS 9037	J. Berry, 8930-1
1	MS 1202	M. Fox, 8920
10	MS 9214	P. D. Hough, 8950
1	MS 9214	T. Kolda, 8950
3	MS 9018	Central Technical Files, 8940-2
1	MS 0899	Technical Library, 4916
1	MS 9021	Technical Communications Department, 8815/Technical Library MS 0899, 4916
1	MS 9021	Technical Communications Department, 8815 For DOE/OSTI